4.Simple Linear Regression

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


dataFrame = pd.read_csv('Age_Income.csv')


age = dataFrame['Age']

income = dataFrame['Income']

num = np.size(age)

mean_age = np.mean(age)

mean_income = np.mean(income)


CD_ageincome = np.sum(income * age) - num * mean_income * mean_age

CD_ageage = np.sum(age * age) - num * mean_age * mean_age


b1 = CD_ageincome / CD_ageage

b0 = mean_income - b1 * mean_age

print("Estimated Coefficients:")

print("b0=", b0, "\nb1=", b1)


plt.scatter(age, income, color="b", marker="2")

response_vec = b0 + b1 * age

plt.plot(age, response_vec, color="r")

plt.xlabel('Age')

plt.ylabel('Income')

plt.show()
```

Dataset:-Age_Income.csv

```
Age,Income
25,25000
23,22000
24,26000
28,29000
34,38600
32,36500
42,41000
55,81000
45,47500
```

5.Multiple Linear Regression

Dataset:-real_estate.csv
https://github.com/subashgandyer/datasets/blob/main/Real%20estate.csv

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, mean_absolute_error


df = pd.read_csv('real_estate.csv')

df.drop('No', inplace=True, axis=1)


print(df.head())

print(df.columns)


sns.scatterplot(x='X4 number of convenience Stores', y='Y house price of unit area', data=df)


X = df.drop('Y house price of unit area', axis=1)

Y = df['Y house price of unit area']


print(X.head())

print(Y.head())


X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=101)


model = LinearRegression()

model.fit(X_train, Y_train)


predictions = model.predict(X_test)


print('Mean Squared Error:', mean_squared_error(Y_test, predictions))

print('Mean Absolute Error:', mean_absolute_error(Y_test, predictions))
```

6) Implementation of Decision tree using sklearn and its parameter tuning

```
import numpy as np

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, classification_report


data = load_iris()

X = data.data

y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

clf = DecisionTreeClassifier(random_state=42)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")

print(classification_report(y_test, y_pred, target_names=data.target_names))

param_grid = {

  'criterion': ['gini', 'entropy'],

  'max_depth': [None, 10, 20, 30],

  'min_samples_split': [2, 5, 10],

  'min_samples_leaf': [1, 2, 4],

  'max_features': [None, 'sqrt', 'log2']

}

grid_search = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42), param_grid=param_grid,
cv=5, n_jobs=-1, verbose=1)

grid_search.fit(X_train, y_train)

print(f"Best Parameters: {grid_search.best_params_}")

print(f"Best Cross-Validation Score: {grid_search.best_score_:.2f}")

best_model = grid_search.best_estimator_

y_pred_tuned = best_model.predict(X_test)

print(f"Tuned Model Accuracy: {accuracy_score(y_test, y_pred_tuned):.2f}")

print(classification_report(y_test, y_pred_tuned, target_names=data.target_names))
```

7) Implementation of KNN using sklearn

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score


data = load_iris()

X = data.data

y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
```

8) Implementation of K-Means Clustering

```
from sklearn.cluster import KMeans

import numpy as np


X = np.array([[1.713, 1.586], [0.180, 1.786], [0.353, 1.240], [0.940, 1.566],

        [1.486, 0.759], [1.266, 1.06], [1.540, 0.419], [0.459, 1.799], [0.773, 0.186]])

Y = np.array([0, 1, 1, 0, 1, 0, 1, 1, 1])

kmeans = KMeans(n_clusters=3, random_state=0).fit(X)

print("The input data is")

print("VAR1 \t VAR2\t CLASS")

for i, val in enumerate(X):

    print(val[0], "\t", val[1], "\t", Y[i])


print("=" * 20)

print("The test data to predict")

test_data = []

VAR1 = float(input("Enter value for VAR1: "))
```

```
VAR2 = float(input("Enter value for VAR2: "))
test_data.append([VAR1, VAR2])
print("=" * 20)
predicted_class = kmeans.predict(test_data)
print("The predicted class is:", predicted_class[0])
```

9) Implementation of Logistic Regression using sklearn

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

iris = load_iris()
X = iris.data[iris.target != 2]
y = iris.target[iris.target != 2]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LogisticRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
print(f'Confusion Matrix:\n{cm}')
print(f'Classification Report:\n{report}')
```

10)Performance analysis of Classification Algorithms on a specific dataset

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_auc_score

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.linear_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier


iris = datasets.load_iris()

X = iris.data

y = iris.target


df = pd.DataFrame(data=np.c_[X, y], columns=iris.feature_names + ['target'])


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


classifiers = {

  'Logistic Regression': LogisticRegression(max_iter=200),

  'Random Forest': RandomForestClassifier(random_state=42),

  'SVM': SVC(probability=True, random_state=42),

  'KNN': KNeighborsClassifier()

}


results = {}
```

```
for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    preds = clf.predict(X_test)

    report = classification_report(y_test, preds, output_dict=True)
    results[name] = {
        'Accuracy': report['accuracy'],
        'Precision': report['weighted avg']['precision'],
        'Recall': report['weighted avg']['recall'],
        'F1 Score': report['weighted avg']['f1-score']
    }

results_df = pd.DataFrame(results).T
print("Classification Metrics:\n", results_df)

plt.figure(figsize=(10, 5))
cm = confusion_matrix(y_test, classifiers['Random Forest'].predict(X_test))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix for Random Forest')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.xticks(np.arange(3), iris.target_names)
plt.yticks(np.arange(3), iris.target_names)
plt.show()
```